Damage Detection and Classification in Wind Turbine's Blades using Automated Visual Inspection

Tiago Miguel da Cunha Cesteiro
tiago.cesteiro@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

July 2020

Abstract

With the rising of awareness regarding global warming, renewable energy sources have been increasingly the focus of the energy industry. According to [11], in 2017, wind power generated the second most electricity amount (7.7%), from all renewable energies (the first one was hydropower with 85.2%). Due to wind erosion and others factors, wind turbine blades are components highly affected during work. In that sense, periodic inspections should be performed to prolong their lifetime. Visual inspections are performed by trained inspectors, however an automatic toll can be implemented to cut down the inspection time and total cost, while also guaranteeing overall coherence in the damages identified. Knowing there are still no automatic solutions for that purpose, this work aims to explore tools for automatic visual damage detection and classification in blades photographies, which were collected from wind turbines farms. For the purpose, two approaches were considered. In a first attempt, an algorithm was developed, using only classic computer vision methods, along with point clustering techniques. Knowing the limitations of this approach, deep neural networks were explored as second option. The purpose of the first approach is to establishing a performance reference for the neural networks. In the end, the approaches were compared for the same set of images, confirming the limitations of the first approach, as well as the potential of deep neural networks to adapt to non-trivial tasks, achieving good generalization and robustness.

Keywords: wind turbines, automated visual inspections, damages detection and classification, computer vision, deep learning

1 Introduction

Wind power was one of the primordial energy source to be used by humans. Since ancient times, various cultures and civilizations were able to convert wind energy into mechanical power. However, wind power was eventually replaced by steam and internal combustion engines, that utilize alternative energy sources, such as coil, fossil fuel, oil, natural gas and nuclear energy. Only a few decades ago, the advance of technology and the increasing of awareness about global warming and environmental pollution has led the energy industry to rely, more and more, on renewable energy sources. Moreover, the costs reduction in this sector made it a reliable and competitive energy source. In this manner, wind turbines have become a more common technology for electricity generation. Due to highly exposure to wind loads, the rotor blades are excessively stressed parts, with high potential to develop damages. Thus, blades are a crucial component to observe during design and operational time of wind turbines.

1.1 Rotor Blades Inspections

To extend rotor blades lifetime, a good maintenance should be conducted by wind farm owners. Nowadays, blade inspections have a range of different techniques, being all of them non-destructive testing. According to [4], this range contains visual inspections, infra-red thermography, ultrasonic and tap testing, digital radiography, acoustic emission, vibration analysis and microwave or terahertz techniques.

Visual inspections is the most commonly used technique, where an expert is attached to ropes and hanged at blades height for a closer look. Some disadvantages arise from this. For instance, the inspector must have a special training and be willing to put his/her life at risk. Also, inspecting all the blades length is time consuming, even for trained experts, and rope access may be difficult in some cases. To overcome such obstacles, the process could be managed with automated visual inspections.

1.2 Automated Visual inspections

Automated Visual inspections (AVI) are accomplished by means of computer vision processes. This field focuses on methods to transfer human vision perception into machines. By combining image processing with other tools, one can develop a computer vision algorithm for a specific AVI case.

AVI started to be used globally in the manufacturing sector in the beginning of 1980s. By guaranteeing controlled image conditions, such as illuminance, scale, rotation and translation, computer vision methods showed to be reliable for real implementations. Since then, AVI expanded to many industries, such as, the automobile, packing, textile and others.

1.3 Problem Formulation

Having all this said, this work will investigate whether AVI is feasible to be implemented in blade inspections of wind turbines. Using a dataset of blades photographs containing several damages, as well as a *.json* file with annotations of detections made by blade inspectors.

In an attempt to unify all 153 labels present in the dataset, a deep observation of the photos was performed, searching for labels with similar damage type. After the aggregation of similar labels and exclusion of irrelevant ones, damages with less than 150 occurrences were excluded. Furthermore, as the data were analysed, some of the bounding boxes showed to be imprecise or the damage type was incorrect, leading for the need to relabel all the dataset. The final dataset, with the correct and consistent labels is described in table 1.

Damage type	Occurrences	Images
Erosion	3948	2475
Peeling	1366	1021
Crack	890	507
Fungi	949	675
Lightning Strike	460	391
Lightning Receptor Damaged	744	743
Total	8357	5750

Table 1: Number of occurrences of each damage in new dataset

1.4 Solutions Proposed

This work explores two different solutions. The first one gives use to classical computer vision methods. Developing this solution implies an individual description of each damage for feature extraction, which is hard-working and highly dependent on the images variety. A second approach employs supervised deep learning. The usage of deep neural networks is being progressively more accessible to everyone, as online frameworks provide libraries with complex network models architectures already built. Also, transfer learning allows models to be pre-trained in huge datasets, accomplishing optimization of the network parameters, which later on can be fine-tuned on a new dataset. This results in a cut down of computational time. However, the application of this solution requires large datasets with ground truth labels, which represents a time consuming effort. Moreover, the network training is a process that takes huge amount of time and computational power. Nonetheless, this solution is less dependent on images variety.

1.5 Objectives and Contributions

The main goal of this work is to explore automated visual inspection tools to detect and classify rotor blades damages. Giving the dataset provided, it was required to apply data preprocessing for overall coherence. As already referred in sec. 1.3, the new dataset was obtained by coupling similar labels, removing data containing few occurrences and relabelling of images damages.

There are still no solutions for this particular problem case, so two approaches were considered, tested and compared. The first approach corresponds to the development of an algorithm using known computer visions techniques to extract individually features of each damage and classify those based on regions properties, such as shapes or density. Also, it is included a segmentation module, where the blade is separated from its background, in an attempt to ease the detection and classification process. The second approach uses an existing deep neural network, which has been previously trained in a large dataset. By means of transfer learning the parameters are fine tuned for the new dataset.

1.6 Outline

This work presents a total of seven sections, including the current one, which already addressed the motivation, along with the problem formulation and proposed solutions. Sec. 2 contains a theoretical review of computer vision techniques required to develop the first solution proposed. Sec. 3 focuses on the theoretical foundations of the second solution. It starts with an introduction to artificial intelligence and deep learning concepts, followed by a choice of the model implemented, which is then described through the remaining of the section. Sec.4 describes and divides the dataset used in the implementations. Sec. 5 the implementation of the proposed solutions, as well as the results obtained with the training dataset. In sec. 6, both solutions are tested and compared for the same dataset. Sec. 7 finishes this document with conclusions and achievements concerning the objectives defined in the previous section. Additionally, suggestions for future work are also included.

2 Computer Vision Background

The literature review shows that is possible to detect damages in images using only classic computer vision techniques. For instance, in [16] a simple algorithm for cracks detection in high-speed steel bar in coil was developed. Also, more complex algorithms, as in [14], exploited the detection of different damages in tiles. However, none of the previous cases presents noisy background as the turbine blades images. To overcome this issue, a preprocessing algorithm based on edges detection and hough transform to remove the background is proposed. Having the blade foreground defined, a combination of morphological operations, edges detection and thresholds are applied to get the RoIs. Finally, the RoIs are classified based on blob analysis, such as size, shape, density and others.

2.1 Spatial Filtering

Spatial filtering is a technique to enhance or remove features of an image, by convolving a filter through it. A filter is a matrix element, F, with dimensions $w \times h \times d$, being w the matrix width, h the height and d the number of channels of the input. This element is convolved across an image I, resulting in a new image I'. The output image is computed as:

$$I'(x,y) = \sum_{i=1+m}^{m} \sum_{j=1+n}^{n} \sum_{k=1}^{d} I(x+i, y+j, k) \cdot F(i,j,k) \quad (1)$$

where $m = \left\lfloor \frac{w}{2} \right\rfloor$ and $n = \left\lfloor \frac{h}{2} \right\rfloor$.

2.2 Binarization

Binarization of grayscale images is an important step towards segmentation and analysis of foreground objects. Many characteristics might be used for the purpose, however the most common is the pixels intensity. Applying a threshold to all pixels may be efficient, yet defining manually a threshold value that better splits foreground into background is not straightforward. The best solution is to apply an automatic threshold criteria, such as Otsu or Adaptive threshold [13] [2].

2.2.1 Otsu Threshold

Otsu method finds a global threshold value that minimizes the intra-class variance of foreground and background. In an iteratively fashion, Otsu method computes different threshold values, followed by an analysis of the classes distribution. The distributions are analysed with the histogram of the grayscale image. Each bin in the histogram corresponds to a different gray value level and the histogram measures the occurrences of those values in the image.

By increasing the threshold for each bin value, the image is divided into background for values bellow the threshold and into foreground for values above it. Having the classes established, the variance of each class is computed as:

$$\sigma^2 = \frac{\sum (v_i - \mu)^2 n_i}{\sum n_i} \tag{2}$$

where v_i and n_i are the value and number of pixels of level i, respectively. Following, the weight of each class, $w = \frac{\sum n_i}{N}$, is calculated, where N is the total number of pixels in the image. The intraclass variance is obtained from an weighted arithmetic mean of both classes variance, $\sigma^2_{global} = w_0 \cdot \sigma_0^2 + w_1 \cdot \sigma_1^2$.

2.2.2 Adaptive Threshold

Adaptive threshold technique computes a threshold for each pixel, based on the local mean intensity of the neighbourhood around it. For a given pixel, p, with a neighbourhood N, the local threshold is applied as:

$$p^* = \begin{cases} 0 & \text{if } p < \text{mean(N)} \cdot s \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

where s is the sensitivity parameter in the range [0 1]. Higher values of sensitivity will threshold more pixels as foreground.

2.3 Morphological Operations

This work uses as baseline dilation and erosion morphological operations. As the names suggest, these operations dilate and erode the foreground pixels using a structuring element. This can be handy to connect or disconnect foreground objects, as well as removing small objects if used in the right sequence. For this last case one requires an opening operation, where the image is eroded in such way that small objects disappear and then is dilated to resize all objects to their original shape. If the sequence order is inverted the operation is called closing.

When dealing with dilation, the structure element origin is passed through each target pixel location and the following condition yields: the target pixel is set to foreground if any of the structuring element coincides with a foreground pixel. Regarding the erosion, the condition yields: the target pixel is set to background if any of the structuring element coincides with a foreground pixel, except in the case of all the structuring elements coinciding with foreground pixels, which the target pixel is left untouched.

2.4 Canny Edge Detection

Canny edge detection, introduced in [3], is a multi step algorithm to detect edges in gray scale images. It starts by reducing the image noise with the application of a gaussian filter. Then, for each pixel the gradient magnitude is computed. Edges are most likely to contain strong gradient magnitudes. Following it, a Non-Maximum Suppression (NMS) is applied to thin out the edges. Finally, hysteresis threshold is performed to the gradients magnitudes.

2.4.1 Gaussian Filter

The gaussian filter, built from the bivariate gaussian distribution, is defined as:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}(\frac{x^2+y^2}{\sigma^2})} \tag{4}$$

where σ is the standard deviation. Since the gaussian distribution is continuous, the kernel values are approximations of it.

2.4.2 Gradient Computation

In image processing it is good practice to use a central difference derivative, which can be translated to a linear filter, [-1 0 1]. The gaussian gradient filter, $\nabla G = (\frac{\partial G}{\partial x}, \frac{\partial G}{\partial y})$, is the output of convolving derivative filters through a gaussian filter. After convolving this gradient filter and obtaining the partial derivatives, g_x and g_y, the gradient magnitude and direction are calculated for each pixel as:

$$|g| = \sqrt{(g_x)^2 + (g_y)^2} \tag{5}$$

$$\angle g = \arctan(\frac{g_y}{g_x}) \tag{6}$$

2.4.3 Non-maximum Suppression

For a given edge point, NMS investigates pixels in the gradient direction neighbourhood and removes those who present smaller gradient magnitude than the current reference pixel.

For a given pixel, q, the gradient direction is propagated in both forward and backwards directions, obtaining the closest points in the gradient direction, r and q. Having this points defined, the pixel q is considered an edge if its magnitude is bigger than both r and p. For more details on this type of non-maximum suppression, recall to [7].

2.4.4 Hysteresis Threshold

In hysteresis threshold, upper and lower thresholds are considered. From the thinned out gradient, any value below the lower threshold is set to 0, while values above the upper threshold are set to 1 and labelled as strong edges. Values that fall between both thresholds are weak edges and are set to 1 if they are connected to a strong edges. The hysteresis threshold outputs the final edges from Canny algorithm in the shape of a binary image.

2.5 Hough Transform

Hough transform was firstly presented in [9], however the popular algorithm was only establish a few years later, in [5]. It is an algorithm designed to detect lines, circles and other curves, knowing their parametric equations.

2.5.1 Lines in Hough Space

A line can be defined in parameter space with the polar equation:

$$\rho = x \cdot \cos(\theta) + y \cdot sin(\theta) \tag{7}$$

where (ρ, θ) are the variables, representing the distance to origin and the angle of the line, respectively. A noticeable property of hough transform is that lines become points in parameter space and points become sinusoidal waves, thus if two points belong to a line in image space, their representation in parameter space will intersect at a point (ρ, θ), which corresponds to the parameters of the line. This is the foundation of hough transform. By observing the parameter space, points with high intersections density are more probable of representing true lines.

2.5.2 Circles in Hough Space

For circles the Hough transform follows the same principle as lines, but with minor changes. A circle in image space can be described by the polar equations:

$$\begin{aligned} x &= a + R \cdot \cos(\theta) \\ y &= b + R \cdot \sin(\theta) \end{aligned} \tag{8}$$

where (a, b) are the circles center coordinates and R is the radius. Considering a constant radius, R, the transformation of points in image to parameter space origins circles. Points with higher accumulated votes are more probable of representing true circles center (a, b).

Nevertheless, in real implementations the radius may be unknown, hence it can not be consider a constant. To include a variant radius in the algorithm, it is necessary to extend the parameter space in the R direction. This means a point in image space will result in a conic surface. In this manner, votes are accumulated for interceptions of conic surfaces.

2.6 Clustering

With clustering algorithms one can group data points into clusters based on specific characteristics, such as similarity between points. For instance, a classic K-means clustering method [12] uses only the centroids distance as similarity, but other metrics may be used, as in the case of Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [6], which clusters points in high density neighborhoods. Additionally, more elaborated versions of K-means can segment images into similar regions based on distance and color of pixels, such as the Simple Linear Iterative Clustering (SLIC) algorithm [1].

2.6.1 SLIC

SLIC algorithm clusters images into approximately equally-sized regions based on the CIELAB color space and Euclidean distances similarity in (x, y) plane. Having this said, the cluster centers, $C_k = [l_k, a_k, b_k, x_k, y_k]$, where the first 3 variables are the values of CIELAB color space, are initialized by a grid of equally spaced intervals. The space between cluster centers depends on the desired number of clusters, K, and on the total number of pixels in the image, N. As for the distance metric, D , SLIC uses a linear combination of CIELAB color space and euclidean distances, d_{lab} and d_{xy}, given as follow:

$$D = d_{lab} + \frac{m}{S} d_{xy} \tag{9}$$

where m is the compactness of clusters and regulates the emphasis that SLIC should give to the (x, y) plane distance when computing the distance metric. Both d_{lab} and d_{xy} are computed as euclidean distances:

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \tag{10}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \tag{11}$$

3 Deep Learning Background

The rising of DNNS in computer vision field was more noticeable since the launch of Imagenet Large Scale Visual Recognition Challenge (ILSVRC), in 2010. As this innovative competition got prestige, many machine learning researchers designed DNNs with peculiar architectures, such as Single Shot multibox Detector(SSD), You Only Look Once (YOLO), R-FCN , R-CNN, Fast R-CNN and Faster R-CNN. In [10] are presented comparisons between these networks regarding different parameters. When fast processing time is required, for instance in real time detections, networks with an architecture like SSD and YOLO show good performances. However, the accuracy of these two is overcome by slower models, like Faster R-CNN. Considering processing time has less relevance, the adopted model was a Faster R-CNN [15].

Concerning the feature extractor implemented in Faster R-CNN, ResNet-101 and Inception ResNet v2 show close results in overall mAP and accuracy, but the second one is much more computational demanding than the first. This last point was the main factor to choose ResNet-101 [8], over the Inception ResNet v2.

3.1 Convolutional Neural Networks Architecture

A typical CNN is composed of three layers: convolutional, pooling and fully-connected layer. Those are described next.

3.1.1 Convolution Layer

Convolutional layers are responsible for generating abstract representations of an input, to be fed into the fully-connected layer, latter on. These representations, usually referred as feature/activation maps, are obtained by convolving a stack of linear filters on the input image. While training a CNN, weights will be adjusted to minimize the output error. In regard to the convolutional layer, these weights are present in the filter. The depth of a convolutional layer is dictated by the number of filters. If one increases the depth, the model becomes more complex, as the amount of weights and feature maps rises.

3.1.2 Pooling Layer

Pooling layers are used to extract the most relevant features in the activation map. This application reduces the dimensionality of the representation and, consequently, the number of parameters needed. Furthermore, its application results in a local translation and minor changes invariance. Pooling can be implemented with different methods, but typically max pooling is chosen in many networks architectures. When the kernel is slid across the activation maps, max pooling will extract the features with highest score.

3.1.3 Fully-Connected Layers

Fully-connected layers are based in multi-layer perceptron models. Each layer is constituted by independent neurons, i.e, there is no connections within a layer. There are three types of layers: input, hidden and output layer. Connections between the first two layers are associated with a weight w_{ij} and from the last two with a weight W_{ij}. Also a bias ,b, is added to hidden and output layers.

A neurons input, z_j, results from the dot product between the weight of the connection, w_{ij} and the output of the previous layer x, plus a bias value, b:

$$z_j(x) = \sum_i w_{ij}x_i + b_j = w^T \cdot x + b \tag{12}$$

A neurons output, known as activation, is computed using the logit, z_j, and an activation function, f:

$$g(x) = f(z(x)) \tag{13}$$

3.1.4 Transfer Learning

Transfer learning is accomplished as follows: firstly, a network is trained from scratch in a large source dataset, obtaining a robust model. Secondly, having the pre-trained model, the parameters are reused to initialize the new network. It is common practice to retrain all the layers of the model.

3.2 Faster R-CNN

Faster R-CNN architecture brings together two main modules. The first one, noted as Regional Proposal Network (RPN), proposes RoIs with various scales and aspect ratios. The second one, known as Fast R-CNN, used as classifier and box regressor for the RoIs. Both modules contain a fully-connected layer, which has two sibling output layers: a bounding box regression and a classification layer. The first is responsible for predicting the coordinates of the bounding-box and the second to output the probability of the predicted box containing an object and of being background. To merge the two modules, a sharing of convolutional feature maps is implemented. In this way, Faster R-CNN learns to generate regional proposals and to classify those, using the same feature maps.

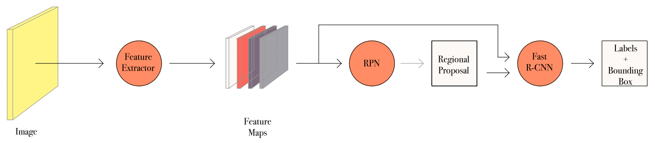


Figure 1: Faster R-CNN, as a unified network

3.3 Activation Functions

Activation functions have the purpose of giving the model non-linearity, which is a requirement when working with complex tasks. In this work there are 3 varieties of activation function: first one for the hidden units, a second one for the classification units, and a third for regression units.

3.3.1 Hidden Layer

The most common function used in units from hidden layer is the ReLU, which is expressed as:

$$f(x) = max(0,x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

3.3.2 Classification Layer

In this work a softmax formulation was used. Softmax is an activation function to obtain a normalized probability of being each class. It is formulated by the following equation:

$$p_k = \sigma(y)_k = \frac{e^{y_k}}{\sum_i^n e^{y_i}}, \qquad k = 1, 2, ..., n \tag{15}$$

where k is the index of each class, n the total number of classes, y the logits vector and p the vector with the probability of being each class, p_k, also known as confidence score. Having this last vector, the desired output of the classifier is the class with higher probability.

3.3.3 Box Regression Layer

This layer receives three input vectors for each anchor: A, G and d. The first vector is composed of the center coordinates, the width and height of the proposed anchors, The second one, similar to the previous one, but regarding the ground-truth box, and the third one is a vector, composed of logits, meaning each element is a linear function of the output from the last hidden layer. To obtain the predicted ground-truth box, P, a regression function formulation was used, as described by the following transformations:

$$\begin{array}{ll} P_x = A_w\ d_x + A_x & P_w = A_w\ e^{d_w} \\ P_y = A_h\ d_y + A_y & P_h = A_h\ e^{d_h} \end{array} \tag{16}$$

3.4 Optimization

Optimization algorithms are responsible for minimizing or maximizing an objective function, which depends on the learnable parameters, and translates the models error. In neural networks, the previous function is referred to as loss function and the learnable parameters are the weights and bias, which are updated at each iteration using gradient-based algorithms. As these parameters are connected in a cause-effect fashion, the algorithm is conjugated with backpropagation.

3.4.1 Gradient Descent

Gradient descent is a learning rule used to minimize the loss function by updating the models parameters in the opposite direction of the loss gradient. The goal is to take steps towards the direction of the downhill surface generated by the loss function, until a local minimum is reached.

Eq. (17) represents the learning rule used in this work. The updated parameter, θ_{t+1}, will depend on the current parameter value, θ_t, on the gradient of the loss function with respect to that parameter, $\frac{\partial L}{\partial \theta_t}$, on the learning rate, η, responsible for controlling the size of step taken between iterations, on the momentum term, α, which determines how much of the previous momentum, v_t will accumulate in the new one, v_{t+1} and on a weight decay, λ, that avoids overfitting of the model, by penalizing the update made in parameters with high values.

$$\begin{aligned} \theta_{t+1} &= \theta_t + v_{t+1} \\ v_{t+1} &= v_t \alpha - \eta \tfrac{\partial L}{\partial \theta_t} - \lambda \eta \theta_t \end{aligned} \tag{17}$$

3.4.2 Loss Function

Faster R-CNN uses a multi-task loss, where the classification and regression loss are coupled in a unified formulation. While the classification output predictions, p_k^P, are obtained by means of (15), the box regression optimization is accomplished by means of normalized coordinates. Arranging the transformations present in (16) to isolate the linear functions, one obtains the normalization of four coordinates. Denoting t^G and t^P as the normalized coordinates of the ground truth and predicted box, respectively, the following is obtained:

$$\begin{array}{llll} t_x^P = \frac{P_x - A_x}{A_w} & t_y^P = \frac{P_y - A_y}{A_h} & t_w^P = \log\left(\frac{P_w}{A_w}\right) & t_h^P = \log\left(\frac{P_h}{A_h}\right) \\ t_x^G = \frac{G_x - A_x}{A_w} & t_y^G = \frac{G_y - A_y}{A_h} & t_w^G = \log\left(\frac{G_w}{A_w}\right) & t_h^G = \log\left(\frac{G_h}{A_h}\right) \end{array} \tag{18}$$

Having the inputs, p and t, established, the multi-task loss, $L(p,t)$, is formulated as follows:

$$L(p,t) = \underbrace{\frac{1}{M}\sum_{m}^{M} L_{cls}(p_m^P, p_m^G)}_{(1)} + \underbrace{\lambda \frac{1}{R}\sum_{m}^{M} p_m^G L_{reg}(t_m^P, t_m^G)}_{(2)} \tag{19}$$

Multi-task loss is accomplished by summing the classification loss (19-1), with the box regression loss (19-2). The variable m is the index of the anchor inside a mini-batch of size M. Also, the loss is normalized by means of M and R, being R the total number of every possible anchor location and aspect-ratio combination. At last, a balancing parameter, λ, is introduced to control the weight of term in (19-2).

Regarding the classification loss formulation, L_{cls}, a log loss is accomplished as:

$$L_{cls}(p_m^P, p_m^G) = -\sum_{n}^{N} p_{m,n}^G \, log(p_{m,n}^P) \tag{20}$$

Concerning the box regression loss, L_{reg}, the following formulation yields:

$$L_{reg}(t_m^P, t_m^G) = \sum_{j \in \{x,y,w,h\}} smooth_{L_1}(t_{m,j}^P - t_{m,j}^G) \tag{21}$$

where $smooth_{L_1}$ is a robust loss function given as:

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{22}$$

3.4.3 Backpropagation

Gradient descent updates the models learnable parameters in the opposite direction of the loss gradient, to minimize it. The term $\frac{\partial L}{\partial \theta_t}$ in the learning rule established (17) represents the sensitivity of the loss function, L, to small changes to parameter, θ. As previously mention, the loss function depends on the ground truth, and on the activation from the neuron of the last layer, a^l. Furthermore, this activation depends on the weight, w^l, on the bias, b^l, and on the activation of the neuron from the previous layer, a^{l-1}. To compute how the weight affects the loss function, chain rule should be applied as next:

$$\frac{\partial L}{\partial w^l} = \frac{\partial L}{\partial a^l}\frac{\partial a^l}{\partial z^l}\frac{\partial z^l}{\partial w^l} = \frac{\partial L}{\partial a^l}\,\sigma'(z^l)\,a^{l-1} \tag{23}$$

where the index l refers to the last layer of the network and $l-1$ to the precedent layer. The term $\frac{\partial L}{\partial a^l}$ represents the sensitivity of the neuron a^l to the loss function and z^l the input of neurons. The derivative of a neuron, a^{l-1}, is computed with chain rule, similar to (23).

4 Dataset

The dataset was already establish, however for implementations purposes some preparation should be made first, as well as announcement of existing constrains. Firstly, the dataset was split into three subsets: train(65%), validation(15%) and test(20%). Table 2 displays the division of damages into these subsets.

Regarding the CVA development, a set of 10 images was selected per damage from the training dataset, where erosion was not included, since it cannot be distinguished by strong features, rather, only by its spacial location in the blade.

	Train (65%)	Validation (15%)	Test CVA (11%)	Test DLA (20%)
Images	3724	940	633	1086
Erosion	2526	633	-	789
Peeling	875	220	271	271
Crack	570	142	178	178
Fungi	607	152	190	190
LS	296	75	89	89
LRD	477	121	146	146

Table 2: Dataset division for implementations

5 Wind Turbine's Blades Damage Detection and Classification Algorithms

5.1 Computer Vision Algorithm

The CVA solution was implemented in *MATLAB* 2017a and is composed of the modules: main (fig. 2), feature extraction (fig. 3), feature classification ((fig. 4, 5, 7, 8 and

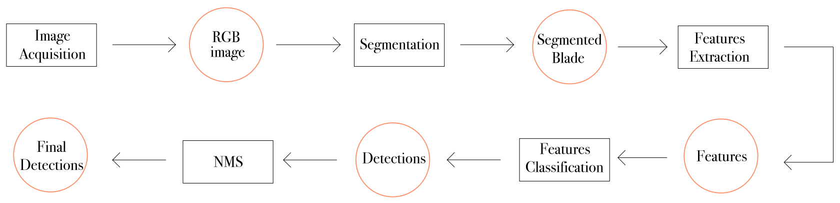


Figure 2: CVA: Main module

9)), segmentation (fig. 6) and NMS. The main module manages all the detection processes. First, it performs image acquisition and then it handles the calling of the remaining modules. Each damage can be identified by the describers: peeling has strong edges, dark area and low spacial density; fungi has strong edges, dark area and high spacial density; crack has strong edges and high eccentricity; Lightning Strike (LS) has a dark and large area; Lightning Receptor Damaged (LRD) has circular shape with less than 2 small circles inside or 1 large circle inside or strong edges in the surrounding area.

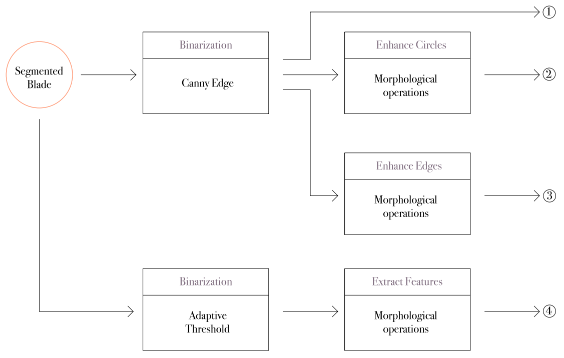
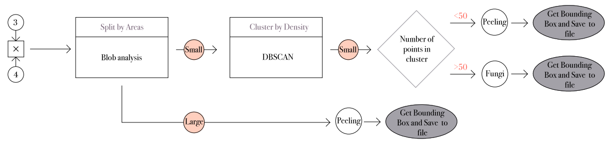
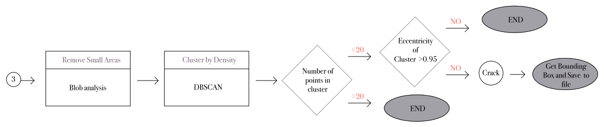


Figure 3: Feature Extraction module



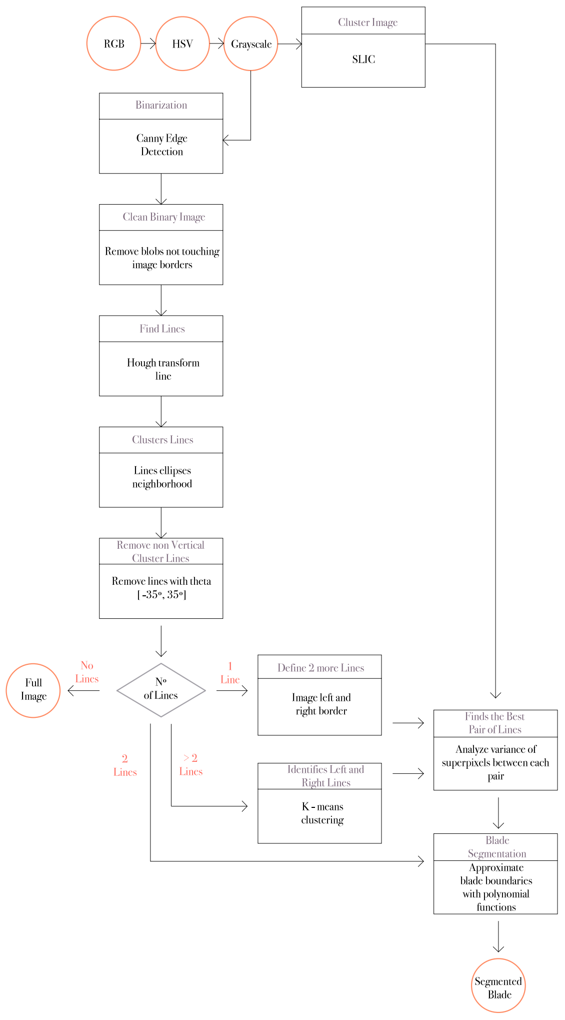
3
4
Split by Areas
Blob analysis
Small
Cluster by Density
DBSCAN
Small
Number of points in cluster
<50
Peeling
Get Bounding Box and Save to file
>50
Fungi
Get Bounding Box and Save to file
Large
Peeling
Get Bounding Box and Save to file

Figure 4: Fungi and peeling classification module



3
Remove Small Areas
Blob analysis
Cluster by Density
DBSCAN
Number of points in cluster
<20
>20
Eccentricity of Cluster >0.95
NO
END
NO
Crack
Get Bounding Box and Save to file
END

Figure 5: Crack classification module



RGB
HSV
Grayscale
Cluster Image
SLIC
Binarization
Canny Edge Detection
Clean Binary Image
Remove blobs not touching image borders
Find Lines
Hough transform line
Clusters Lines
Lines ellipses neighborhood
Remove non Vertical Cluster Lines
Remove lines with theta [-35º, 35º]
Nº of Lines
No Lines
Full Image
1 Line
Define 2 more Lines
Image left and right border
2 Lines
> 2 Lines
Identifies Left and Right Lines
K - means clustering
Finds the Best Pair of Lines
Analyze variance of superpixels between each pair
Blade Segmentation
Approximate blade boundaries with polynomial functions
Segmented Blade

Figure 6: Segmentation module

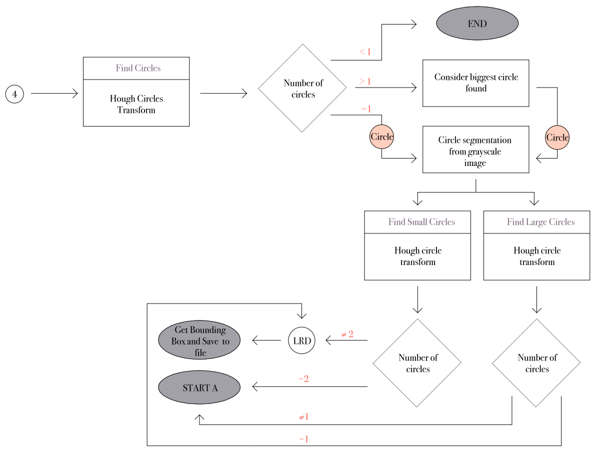


Figure 7: LRD classification module

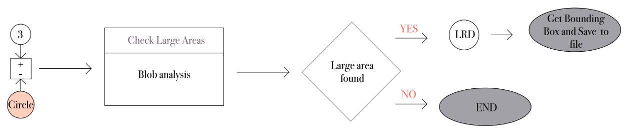


Figure 8: Start A from LRD classification module

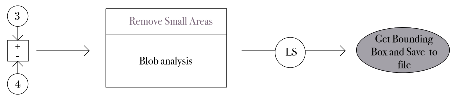


Figure 9: LS classification module

5.1.1 Non-Maximum Suppression

Since each damage methodology did not take into consideration others damages, it is possible for those to mislead the CVA for incorrect or redundant detections. In that sense, a non-maximum suppression was applied. The following conditions and consequent actions, were considered sufficient enough to improve the precision, while not ruining recall: Equal label + $IoU_{min} = 1 \implies$ smaller area is fully within the biggest one. Remove small area; Equal label + $\frac{1}{3} < IoU_{min} < 1 \implies$ merge both bounding boxes to cover all the union area; LS vs peeling/crack + $IoU_{min} > \frac{1}{3} \implies$ remove peeling/crack; LRD vs peeling/crack $+IoU_{min} > \frac{1}{3} \implies$ remove peeling/crack; Crack vs peeling + $IoU_{min} > \frac{1}{3} \implies$ remove peeling.

5.1.2 Results

In order to validate this solution, the CVA was ran for the training dataset already described in section 4. Table 3 shows the recall and precision of each damage using an IoU of $\frac{1}{3}$ (for more details on the metrics used, recall to the full version of this work). Moreover, it is also displayed the results obtained without using NMS. First, observing the recall values, the peeling presents the lowest value, which translates a poor detection of the ground truth cases. On the other hand, the remaining damages recall show that at least half of the ground truths were detected, except for the LS case, which detected correctly all the ground truth cases. The application of NMS improved slightly the precision of peeling, crack and LS damages, while maintaining the recall values. Nonetheless, the peeling precision is still pretty low, when compared to other damages, which means the algorithm is producing a large amount of false positives predictions. To understand these results and find the algorithm flaws, one should visually analyse the results.

Damage type	*Recall*(%)		*Precision*(%)	
	No NMS	NMS	No NMS	NMS
Peeling	22.22	22.22	2.21	4
Crack	51.85	51.85	18.42	22.95
Fungi	56.25	56.25	45	45
LS	**100**	**100**	31.25	43.48
LRD	50	50	**62.50**	**62.50**

Table 3: Average precision for test dataset

5.2 Deep Learning Algorithm

This solution was implemented in TensorFlow, using a high-level Application Programming Interface (API).

5.2.1 Hyperparameters Selection

The hyperparameters were reused from the pre-training process. Knowing these hyperparameters achieved state-of-art results in the COCO dataset, the chances of resulting in the new dataset are huge. Some hyperparameters were already established, while others were not. Table 4 summarizes all hyperparameters.

5.2.2 Input Resizing

Faster R-CNN has the peculiarity of working in datasets composed of images with different sizes. To do it, Faster R-CNN resizes images so that the shorter side has a maximum of 600 pixels, while maintaining, if possible, the aspect ratio. Also, the longest side is restricted to a maximum of 1024 pixels.

5.2.3 Data augmentation

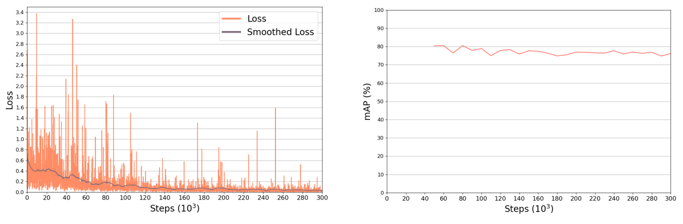
During the training stage, three operations for data augmentation were randomly applied: horizontal flip, brightness adjustment and constract adjustment. The combination of these operations decreases the chances of Faster R-CNN overfitting with the training dataset.

5.2.4 Results

The training process was set to a total of 300K steps, during which the validation dataset was evaluated at every 10K steps. Concerning the transfer learning process, no layers were frozen during training.

The loss was stored in memory every 100 steps and is displayed in figure 10a. A smoothed loss was also computed, to better visualize the decrease in loss values. Even though the loss presents many peak fluctuations, the magnitude and frequency decrease as the training evolves. Regarding figure 10a, since the loss globally decreases, it is conclusive that, while training, the model learned how to adapt to the training dataset. Figure 10b gathers the mAP of the evaluations performed during training, in the range 50K to 300K steps.

In case the model overfitted with the training dataset, the validation mAP would decrease in a conclusive way, but that is not the case. Instead, it is observed small variations of the mAP. Nonetheless, the model seems to perform better with the validation dataset before overpassing 80K steps, which could mean the model is overfitting with the training dataset after those steps (since the loss continues to decrease). For this reason the model trained for 80K steps was selected for later testing.



(a) (b)

Figure 10: (a) - Loss value every 100 training step, (b) - Evalutation of validation dataset every 10k steps.

Hyperparameters	
New layers weight initialization	zero-mean Gaussian distribution with standard deviation 0.01
Hidden Layer: activation function	ReLU
Classification Layer: activation function	Softmax
Regression Layer: activation function	Regression
Gradient Descent with Backpropagation	learning rate η: 0.0003, momentum α: 0.9, mini-batch size M: 256, balancing parameter λ: 10

Table 4: Hyperparameters of Faster R-CNN, as in the original paper [15]

6 Analysis and Comparison of Results

6.1 CVA Testing Results

The CVA solution was run for the test dataset defined in sec. 4. Comparing these results with the ones obtained with the training dataset (table 3), the algorithm performed worst for the test dataset. This could be explained knowing the CVA was built on few images, relatively to the test dataset size. In that sense, even though the calibrated parameters showed feasible results during the development, the same can not be ensured for testing. The fungi algorithm seems to be the more robust, presenting the lowest performance drops, when comparing the development stage with the test. On the other hand, the LS algorithm, which obtained a 100% recall and 43.48% precision on the development stage, dropped significantly its performance. Concerning the peeling and crack damages, the bad results can be related to a lack of generalization in the algorithms. Peeling and cracks have the most irregular shapes and sizes, so the usage of classical computer vision techniques is not enough to ensure appropriate results. The LRD class achieved relatively good results, yet it should be considered that all datasets presented few occurrences of Lightning receptors in good conditions. This means the results may not be conclusive, as the algorithm could be detecting any lightning receptor, instead of catching damaged ones.

Damage type	Recall(%)	Precision(%)
Peeling	15.13	2.66
Crack	26.14	8.30
Fungi	**44.74**	38.46
LS	23.60	6.86
LRD	36.99	**41.86**

Table 5: CVA performance for test dataset

6.1.1 DLA Testing Results

The model proposed in section 5.2.4 was evaluated using the test dataset described in section 4, and was set to output 300 detections, choosing those with higher confidence score. Table 6 presents the Average Precision per class, while table 7 shows the mean Average Precision (mAP) for the test and validation datasets. Once again, the LRD stands out for having the highest result. As explained in section 4, the class erosion was only evaluated for the deep learning solution, as an attempt to confirm whether the network can distinguish a class which depends on the natural context where it is inserted. With a look at the average precision per class, the erosion presents a good value of Average Precision (AP), meaning the previous hypothesis is confirmed. Nevertheless, this class contains the most amount of occurrences, which could be beneficial in the training stage, achieving better results. Regarding table 7, the performance of the testing stage was worst than the validation performed during training (see fig. 10b).

Damage type	Average Precision(%)
Erosion	70.34
Peeling	62.92
Crack	72.57
Fungi	68.94
LS	66.99
LRD	**93.47**

Table 6: Average precision for test dataset, for the DLA

	Validation	Test
mAP(%)	**80.54**	72.54

Table 7: Comparison between mAP in validation and test datasets, for the DLA

6.2 Performance Comparison

The comparison of the solutions proposed cannot be done directly using mAP, since the CVA does not output a confidence score. It was established the following comparison: precision and recall are computed for CVA and checked whether the corresponding point would fall inside or outside the Faster R-CNN average precision area. In case it falls inside, the CVA is considered to have lower performance. Figure 11 demonstrates exactly this comparison. The results conclude that the CVA solution is outperformed by Faster R-CNN. Both peeling, crack and LS detections from CVA show really low precision values. The classes fungi and LRD show slightly larger values of recall and precision, but are also outperformed by Faster R-CNN.

7 Conclusions

Automatic visual inspection of wind turbines blades is not a trivial task. First, concerning the damage classification, there is still no standard labelling system, as it will depend on the inspector, country and training. Second, the photographs themselves contain external factors such as illumination, which can not be controlled and will influence the results. In sec. 5.1, limitations were found in the development of the first solution. Besides the external factors already referred, the manual calibration of parameters was a large workload and showed good results in a design stage, even though during testing stage its performance dropped.In sec. 5.2 transfer learning was applied in a Faster R-CNN with a Resnet-101 feature extractor. The hyperparameters were reused from the pre-training

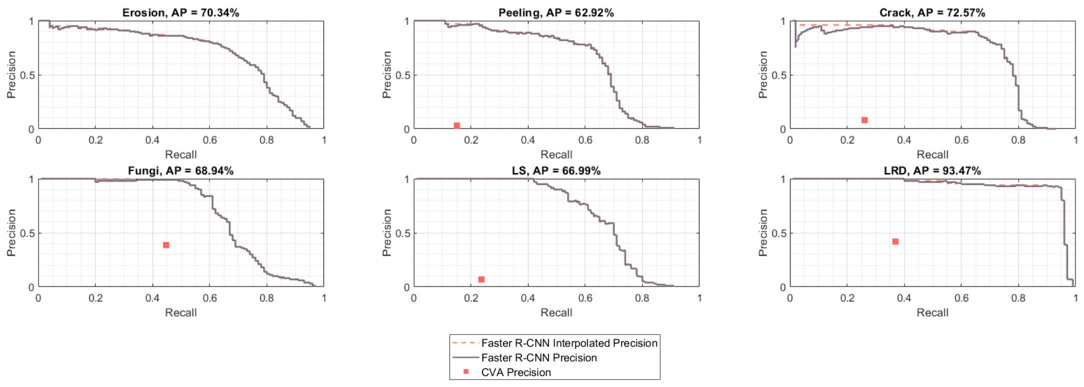


Figure 11: CVA and DLA performance comparison

process. In seek for better results, the hyperparameters sensitivity should have been evaluated, but that would demand a larger amount of computation time. Since the loss converged to zero and the validation performance stagnated, its conclusive the model adapted to the training dataset, while not overfitting with it. The model trained for 80K steps was chosen for testing, since it had the highest mAP in the validation. The testing confirmed that erosion can be detected in the same manner as other damages, meaning the model learned the spatial context of damages. As the LRD class achieved a surprisingly high AP, it is presumed that this was caused by the lack of examples of lightning receptors in good conditions. Even though data augmentation was performed during training, the dataset should be larger to improve the results. In sec. 6, the solutions were compared for the same dataset, which made conclusive that Faster R-CNN performance was much higher than the CVA solution, achieving better generalization in all the classes.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012. 4

[2] D. Bradley and G. Roth. Adaptive thresholding using the integral image. *Journal of graphics tools*, 12(2):13–21, 2007. 3

[3] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:679–698, 1986. 3

[4] T. Chady, R. Sikora, P. Lopato, G. Psuj, B. Szymanik, K. Balasubramaniam, and P. Rajagopal. Wind turbine blades inspection techniques. *Organ*, 5:16, 2016. 1

[5] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. Technical report, Sri International Menlo Park Ca Artificial Intelligence Center, 1971. 4

[6] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996. 4

[7] D. A. Forsyth and J. Ponce. *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002. 3

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 4

[9] P. V. Hough. Method and means for recognizing complex patterns, Dec. 18 1962. US Patent 3,069,654. 4

[10] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors (2016). *arXiv preprint arXiv:1611.10012*. 4

[11] IRENA. Renewable energy statistics 2019 - electricity generation. `https://www.irena.org/Statistics/View-Data-by-Topic/Capacity-and-Generation/Statistics-Time-Series`, 2020. 1

[12] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967. 4

[13] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. 3

[14] G. Rahaman, M. Hossain, et al. Automatic defect detection and classification technique from image: a special case using ceramic tiles. *arXiv preprint arXiv:0906.3770*, 2009. 2

[15] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015. 4, 9

[16] J. P. Yun, Y. Park, B. Seo, S. W. Kim, S. H. Choi, C. H. Park, H. M. Bae, and H. W. Hwang. Development of real-time defect detection algorithm for high-speed steel bar in coil (bic). In *2006 SICE-ICASE International Joint Conference*, pages 2495–2498. IEEE, 2006. 2